This document is based on the group meeting from 2022-09-24, which can be found in the meetings folder. A full comparison and record of discussion for alternative tech stacks can be found there.

Frontend: React

Description:

React is a free and open-source JavaScript framework for building UI components and web applications. It's maintained by Meta, and is currently the most popular frontend framework used in the industry, with companies such as Netflix, and Airbnb reportedly incorporating it in their tech stack.

Reasons:

1. **Experience**. Everyone on the team has various degrees of experience working with React, but very little experience working with other frontend frameworks such as Angular and Vue. This means if we were to choose another framework, we would have to dedicate more time to learning and less time for design and implementation.
2. **Division of work**. Since React is component based, we figure it would be easy to divide up the work should there be a need for us to build multiple components.
3. **Easy to integrate with external libraries**. We found there are some libraries such as Material UI, Recharts, and Next.js that can be easily integrated into our project. These libraries can cut down on the amount of time it takes to build components, and provide tools to do server-side rendering with, should the loading speed of our website be a concern.

Backend: Node + Express and SQLite

We will omit the description and reasons for SQLite as it was strongly recommended from the professor.

Description:

Node.js is an open source runtime environment and library for executing server-side code, and Express.js is a Node.js framework that provides the tools needed to build an API.

Reasons:

1. **Less development overhead**. Everyone on the team has experience with JavaScript, and having the backend use the same language as the frontend means potentially less cost with context switch.
2. **Easily integrate packages**. Node.js comes with npm, which is a great package manager that we can use to install libraries for both backend and frontend. Express also has some documentation on database-integration for SQLite.

Container

Description:

We are containerizing our application using Docker. Our one-click setup script supports the complete install of Docker based on the official docker-install script.

Reasons:

1. **Utility**. Docker allows for the setup of a microservices architecture, basically letting us split up our application into a collection of smaller services.
2. **Compatibility.** Developers love Docker because the containerized version of the application can be compatible with whatever operating system the parent machine runs on, due to its isolation from other processes on the host machine.
3. **Scalability**. Docker is directly scalable with complementing frameworks like Kubernetes to allow hosting that has low to no downtime solutions, leveraging distributed computing to bring our application to consumers.

CI / CD: Github Action

Description:

Github Action is a platform-native automation tool, GitHub Action has evolved to give developers powerful automation and CI/CD capabilities right next to our code in GitHub. Compared with other automation or CI/CD tools, GitHub Actions offers native capabilities right in our GitHub flow.

Reasons:

1. **Ease of use**. Since Github Action automates everything within the Github flow, developers can build a CI/CD pipeline with just a few clicks unlike most CI pipelines

which require you to set up complicated software like Jenkins or be using a specific cloud service like AWS CodePipeline.

2. **Setup and Maintenance**. With other CI pipelines, you will be running it on a custom server. This means that you will have to maintain the server continuously. However, Github Actions provides you with free runners you can use to perform your CI/CD operations. These runners are owned and maintained by Github but you can add self-hosted runners as well.